

Paper 001-31

Service-Oriented Architectures – Going from Buzz to Business

Dan Jahn, SAS Institute Inc., Cary, NC

ABSTRACT

The big promise of Service-Oriented Architectures is their ability to solve business integration issues. This promise is based on applications coded to a common set of standards. Any application that can communicate with the standard can participate in the architecture. This paper examines how SAS applications can call and be called through Web Service standards including XML, SOAP, WSDL, WS-Security, and HTTP. Demonstrations include exposing a SAS program (Stored Process) in a Web Service.

INTRODUCTION

A Service-Oriented Architecture (SOA, pronounced SEW-AH) is just the latest term for cross-application integration. The success of SOA is based on the fact that it makes simple things simple. If you don't care about transactions, authentication, encryption, scalability, attachments, or events, then you don't have to use them. However, if you are developing an application that needs these things, you can make them work. Pick-and-choose what you want, and use only those elements that make sense for your application. The basis of SOA is the message. Anything that can send and receive a message can be considered a Service in a Service-Oriented Architecture. SOA doesn't mandate how that message is passed. The most common mechanism for passing messages in a SOA is HTTP; but message queues, files, e-mail, and FTP are also valid transports. Although sockets themselves are the basis for many transports, sockets are usually not considered valid transports for a SOA because sockets require specialized logic to retrieve messages.

BREAKING THROUGH THE BUZZ

"Service-Oriented Architecture" is a buzzword. As such, you can assign it just about whatever meaning you want. Everyone agrees that SOA implies integration. A 'good' Service-Oriented Architecture is an architecture in which you can easily modify one Service without affecting the other Services. If you are forced to re-compile one Service because you changed another Service, then you don't have a SOA. Certainly, there are times, as you are developing your architecture and its Services that you'll have to change the format of the message that passes between Services. However, after you put your Service into production use, you should avoid changing those interfaces. Instead, you should create new interfaces so that you avoid breaking consumers of your old interface.

Note: In a SOA, *interface* means the format of the message.

The key elements to keep in mind about SOAs are: Integration and Messages. Your goal is to integrate applications, and you do that by using messages.

EXPLORING THE ALTERNATIVES

SAS provides many alternatives for integrating SAS with other applications. These alternatives include:

- Web Services
- COM interfaces
- Java classes
- CORBA bindings
- message queues
- directory services
- database integration (via SAS/ACCESS®)
- HTTP (via SAS/IntrNet®)
- SAS (via SAS/CONNECT®)
- fileref access methods (file, URL, e-mail, pipe, socket, FTP, and so on)

It's important to realize that a SOA isn't necessarily the best choice for every situation. Having some logic on the other end of the communication channel can improve performance. Therefore, there are two major reasons for NOT using a Web Service: Web Services are stateless, and Web Services are inefficient for large amounts of data.

STATELESSNESS

SAS® Enterprise Guide® is a great example of an application that works better because SAS maintains state for it. When you submit SAS code via SAS Enterprise Guide, you know that the effect of that SAS code will be maintained for however long you use the server. In SAS Enterprise Guide, you make a connection to a SAS Server and hold that connection for as long as you are using it. If the first thing you do is submit a LIBNAME statement, then you know you can use that LIBNAME until you stop using the server, which might be a few seconds, days, or weeks. You can make multiple calls to that server and that LIBNAME statement will be in effect until you un-assign it or disconnect from the server. There are many pieces that comprise state in SAS, and all this state is wrapped up by what is called a SAS *Workspace*. The major pieces of state include but are not limited to: LIBNAME assignments, fileref assignments, contents of the temporary WORK library, and the current location in the submitted code.

Web Services are stateless. You make a request. You get a response. When you make the next request, the Service is re-set back to where it was before your first request. Of course, any changes that are intentionally persisted by the Service (such as adding database records) are not re-set.

It is possible to implement your own conversational semantics for a Web Service (for example, adding a session ID parameter to every call). However, this is not provided for you, and you would have to write the Service to specifically save this state. You would also have to solve the problem of when it's safe to release the state. If you have an application that requires SAS to save more state between calls than writing records to a database, then you should consider using something other than Web Services. This might be a good place to use either the Java or the COM based bindings into SAS. These IOM connections remain connected to SAS as long as the client wants to hold the connection. (IOM is the technology that SAS uses to enable Java and COM client access.) SAS Enterprise Guide uses IOM to communicate with SAS.

Note: There is currently a proposal (WS-Conversation) for enabling conversational Web Services in a standard way, which would make it easier to create stateful Web Services.

PERFORMANCE

There is a lot of overhead when using XML. First, the requestor must encode the message data from its native binary format to the all-string XML. Then, the data is sent across the wire (usually, over HTTP). However, there's a LOT more data in an XML representation than there is in binary format, so there's a lot more going across the wire. Next, the receiver must validate and parse the XML back into a native binary representation and perform processing. This entire process must be repeated in order to send a response back to the client.

The graphs in Figure 1 show that getting large amounts of data from SAS through the binary IOM interface is much faster than using the Web Service XML interface. The more data you have, the bigger the difference will be between the binary and XML interfaces.

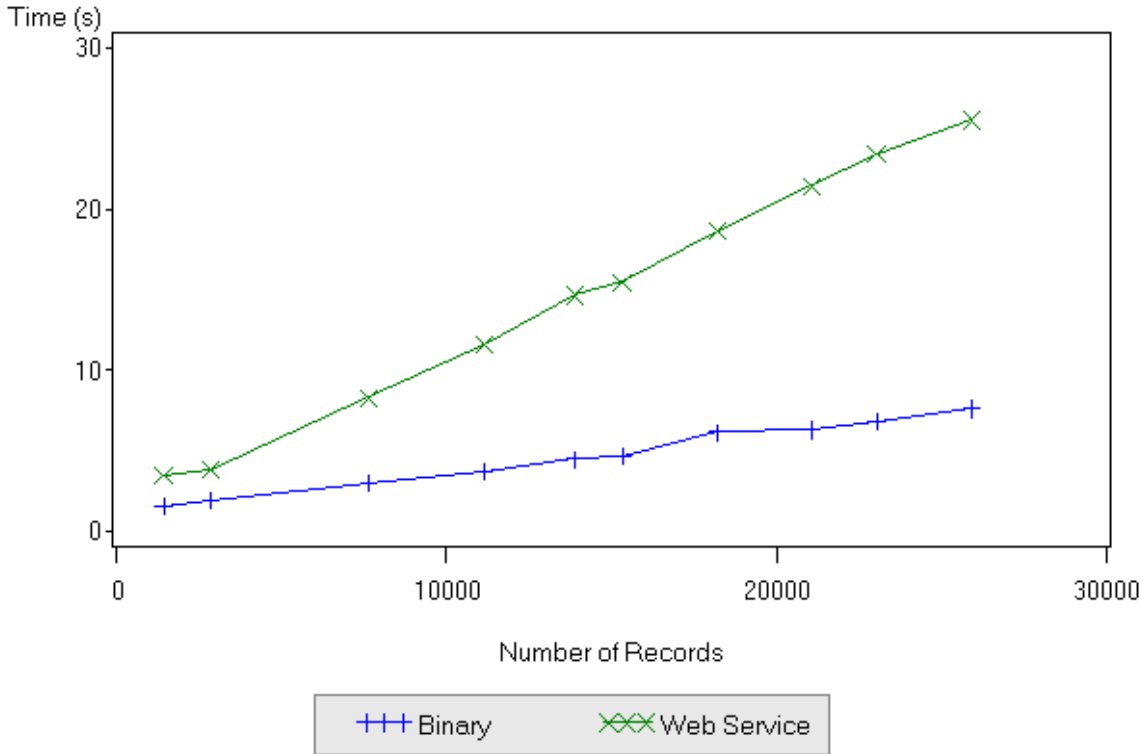


Figure 1. Comparison of Time It Takes to Move Data When Using Binary and Web Service Interfaces

The bar charts in Figure 2 show that the bandwidth that's required for XML is much greater than the bandwidth that's required for IOM.

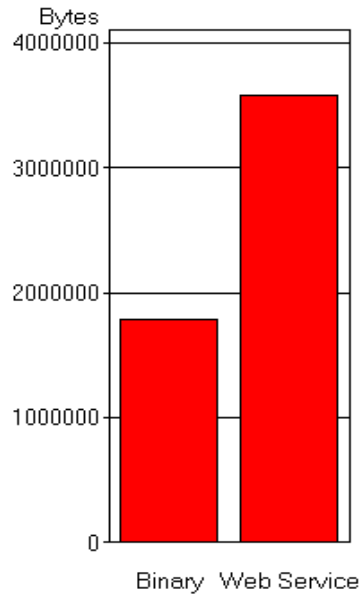


Figure 2. Bandwidth Required by IOM and XML for Approximately 14,000 records

Having standards for binary XML is probably not too far in the future but, until that time, you should realize the performance penalty for using XML. Even when there are binary standards, you should try to have your data on the server to minimize the amount of data going across the connection. Local disk access will continue to be faster than network access, for some time.

STORED PROCESSES

SAS Stored Processes consists of SAS code and metadata that describe how to run the code (including the inputs and output of that code). SAS Stored Processes is an excellent tool for a Service-Oriented Architecture. The Stored Process is the fundamental building block for presenting SAS as a Service that can be used as part of a SOA. Putting it another way, the benefits of a SOA are in line with the benefits of using SAS Stored Processes. The Service is the Stored Process.

SAS provides the following tools for creating Stored Processes:

- SAS DMS with SAS® Management Console
- SAS® Data Integration Studio (previously known as SAS® ETL Studio)
- SAS® Enterprise Guide®
- SAS® Forecast Server

SAS provides the following tools for calling Stored Processes:

- SAS® Information Delivery Portal
- SAS® Stored Processes Web App
- SAS® Web Report Studio
- SAS® Add-In for Microsoft Office
- Java and COM interfaces for custom code
- SAS® BI Web Services

Here are some of the qualities of SAS Stored Processes that make them effective Services (see “References” (Box, 2004) at the end of this paper to compare these to the four Tenets of a Service-Oriented Architecture):

- There is a clear boundary between the caller of the Stored Process and the SAS code. The SAS code is on the server and is not provided by nor can it be modified by the caller of the Stored Process.
- The SAS code can be modified without having to re-compile the caller.
- The input and output of the Stored Process are in metadata, and they are used as the basis of integration (that is, what you need to know to use the Stored Process).

Note: You must NOT modify the metadata of the Stored Process after it's in use. Modification would change the format of the message. This metadata forms a contract between the caller of a Stored Process and the author of the Stored Process. This contract describes what can be accepted as input to the Stored Process, and what will be provided as output from the Stored Process.

- Policies about who can call the Stored Process are managed in metadata.

The metadata to run a Stored Process is very similar to the metadata that describes how to run a Web Service (the Web Service metadata being in a Web Services Description Language (WSDL, pronounced WIZZ-DUL) file). Remember that an important part of a Service-Oriented Architecture is the Services communicating via messages. To communicate with Stored Processes through messages, you can use SAS BI Web Services. The Stored Process is the foundation of SAS BI Web Services.

WEB SERVICES

Web Services make it really easy to follow the principles of a Service-Oriented Architecture, and Web Services are currently the most common implementation tool for Service-Oriented Architectures. SAS BI Web Services, which enables Stored Processes to be exposed as Web Services, started shipping in SAS 9.1. In SAS 9.2, this ability will be enhanced to make it easier for clients to call.

SAS BI WEB SERVICES IN SAS 9.1

SAS BI Web Services is currently available as part of SAS Integration Technologies software. To use SAS BI Web Services with your Stored Processes, you must perform the following tasks:

- 1) Install SAS BI Web Services. SAS BI Web Services requires the SAS Metadata Server, a Stored Process Server, and a Web container that can be either a Java 2 Enterprise Edition (J2EE) such as Tomcat or WebLogic or .NET (IIS). SAS provides a different mid-tier install for each of these environments. You choose which to install based on which you would rather administer and manage. The client calling the Web Service will not see a difference between calling the Java or the .NET mid-tier.
- 2) Write the SAS code. The SAS code is responsible for reading input streams (usually using the XML LIBNAME Engine); reading input parameters, which are passed in as macro variables; and generating XML output (again, using the XML LIBNAME Engine). The SAS code must be stored in a file that is accessible from the Stored Process Server.
- 3) Define the metadata for your Stored Process. Be sure to add the 'XMLA Web Service' keyword to the metadata for your Stored Process. This metadata includes the name of the machine where the Stored Process Server is running, the names of the input streams, and the names of the input parameters.
- 4) Tell your clients to call it!

A WSDL file describes in XML, to a client of a Web Service, how to call the Web Service. The WSDL contains the URL of the Web Service, the set of operations (methods) exposed through the Web Service, and the types (schema of the XML that's used in the messages to communicate with the Web Service.) The 9.1 release of SAS BI Web Services uses a fixed WSDL file to describe the input and output of the Web Service. The WSDL file does not change based on what Stored Processes can be called through the Web Service. The WSDL file describes two operations: **Discover** and **Execute**.

The **Discover** operation enables the client to get a list of the Stored Processes that are available and the parameters that each Stored Process takes. Most client application developers will call **Discover** when they are designing the client application. They probably won't call **Discover** from their production application.

The **Execute** operation enables the synchronous execution of a Stored Process. In order to accommodate any and all possible types of parameters, the types section of the WSDL file describes the input and output of the **Execute** operation using the XMLSchema '**any**' element. The `xs:any` element indicates that any XML element is allowed, and it allows a great deal of flexibility in what you can do with the **Execute** method. The big drawback to `xs:any` is that most client tools use the types to put together the XML that is needed to call a Web Service. Without the detailed types, client application developers are left with using **Discover** and manually putting together the XML to invoke a Stored Process via SAS BI Web Services. Manually creating XML isn't that difficult. Here's an example of XML to call a Stored Process called GETTABLE, which takes as input a single parameter—the name of the table to be returned.

```
<Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
  <Command>

    <!--This comes from SAS BI Web Services Explorer-->
    <StoredProcess name="/BIP Tree/sasrepository/gettable">
      <Parameter name="tablename">sashelp.class</Parameter>
    </StoredProcess>
    <!--End from SAS BI Web Services Explorer-->

  </Command>
  <Properties>
    <PropertyList>
      <DataSourceInfo>Provider=SASSPS,</DataSourceInfo>
      <Content>Data</Content>
    </PropertyList>
  </Properties>
</Execute>
```

The preceding example assumes that you have a Stored Process named GETTABLE that takes the single input parameter TABLENAME, which is already defined in your metadata. This sample XML does not include the SOAP envelope or any namespace definitions.

The typical workflow for using Web Services involves the phases Development and Run Time. Usually, the development of a client that calls Web Services involves generating proxy classes that provide a nice parameterized function to call a Web Service. The proxy function actually puts together the XML to invoke the Web Service operation. In Java, you would use WSDL2Java. In .NET, you would use WSDL.exe. For SAS BI Web Services, the proxy generator will create the two methods **Discover** and **Execute**. Here's the function prototype that a proxy generator created for **Execute** (after cleaning up a few extra attributes that the proxy generator added).

```
public System.Xml.XmlElement Execute([System.Xml.XmlElement Command,
System.Xml.XmlElement Properties)
```

What do you pass in for the Command parameter? For the GETTABLE method, you pass in the following XML:

```
<StoredProcess name="/BIP Tree/sasrepository/gettable">
  <Parameter name="tablename">sashelp.class</Parameter>
</StoredProcess>
```

For the Properties parameter, you pass in the following XML:

```
<PropertyList xmlns="urn:schemas-microsoft-com:xml-analysis">
  <DataSourceInfo>Provider=SASSPS</DataSourceInfo>
  <Content>Data</Content>
</PropertyList>
```

To determine the XML for the Command parameter, call the **Discover** method with the requestType parameter set to STOREDPROCESS_PARAMETERS, and you'll get back XML that describes how to invoke the **Execute** method. Then, at run time, you would normally just call the **Execute** method by passing in the XML that was described when you called **Discover**.

To make it easier to use SAS BI Web Services, SAS released two helper tools for developers: the SAS BI Web Services Explorer and the SAS BI Web Services Wizard (for Visual Studio .NET © Microsoft). The Explorer is a Web-based tool that enables you to browse and execute SAS Web Services; it is also a great tool for learning how SAS BI Web Services behave. Developers also find the Explorer useful for getting the XML that's needed to invoke a Stored Process (the XML shown earlier came from the SAS BI Web Services Explorer). The SAS BI Web Services Wizard is an alternative to using WSDL.exe and will generate either C# or VB.NET code that calls the **Execute** method for you. The Wizard works in a way that is similar to the way that Visual Studio works with WSDL and in generating proxies for Web Services. The Wizard also generates comments that are displayed by IntelliSense, which results in a very rich development experience. (See the SAS Downloads page to get the SAS BI Web Services Explorer and Wizard.)

The Web Service implementation uses a multi-tier architecture. (See Figure 3.)

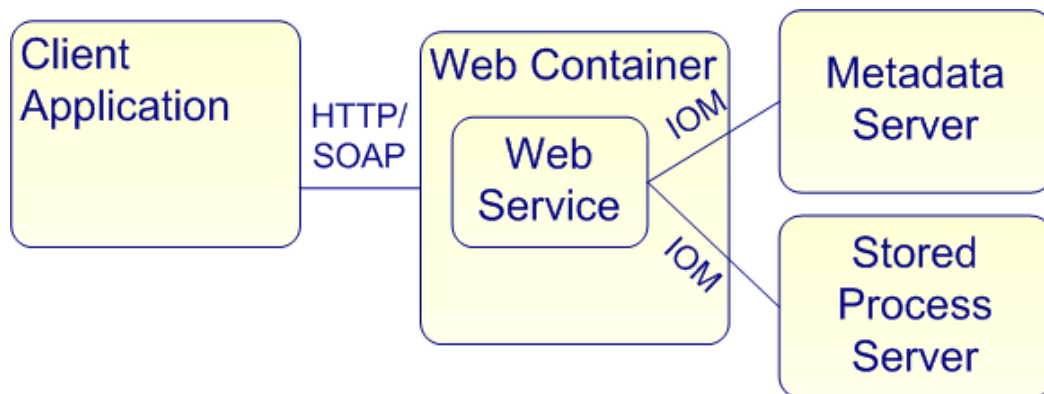


Figure 3. Multi-Tier Architecture for Web Service

When the client calls **Discover**, the mid-tier Web Container receives the request and calls the Metadata Server to get the Stored Process metadata. **Discover** will return all Stored Processes that have the 'XMLA Web Service' keyword. When the client calls **Execute**, the mid-tier Web Container calls the Metadata Server to validate the parameters that are passed-in and to find the Stored Process Server where the job should run. Then, the job is run on the Stored Process Server, and the results are returned to the client. This same multi-tier architecture is used in SAS 9.2 BI Web Services.

SAS BI WEB SERVICES IN SAS 9.2

SAS 9.2 will continue to support the Discover/Execute Web Service, and adds an exciting, new, dynamic Web Service generation capability. SAS BI Web Services will generate a new WSDL document when the administrator deploys the Web Service. Here are the new steps for using SAS 9.2 BI Web Services.

- 1) Installation will be the same as in SAS 9.1. However, when you install SAS 9.2 BI Web Services, you are actually installing two Web Services: the same Web Service that is in SAS 9.1 and the new WebServiceMaker Web Service.
- 2) Write the SAS code.
- 3) Define the metadata for your Stored Process. You no longer need to include the 'XMLA Web Service' keyword in order for your Stored Process to work with the new WebServiceMaker Web Service.
- 4) Using the BI Manager (which is a SAS Management Console plug-in), you can select a set of Stored Processes and deploy them to the WebServiceMaker. The WebServiceMaker will generate a new Web Service that contains one operation for each Stored Process that you selected.
- 5) Create clients to call it!

The newly generated Web Service's WSDL document will contain all the information that you previously got through the **Discover** method. It is now possible to create type definitions for your Web Service that do not use `xs:any`. You can still end up with `xs:any` when you do not define the schema associated with an input or output stream. `xs:any` can be useful when your Web Service can generate output where the schema of the output data varies. Having all the type information in the WSDL is better suited to most client applications because there are numerous standard tools that can generate code to call a Web Service based on WSDL.

Here's the request for the GETTABLE Stored Process that we looked at previously. Recall that it takes as input a single parameter—the name of the table to be returned.

```
<gettable>
  <tablename>sashelp.class</tablename>
</gettable>
```

Making the WSDL more specific to the actual parameters instead of having a generic interface enables you to simplify the request XML. Making the WSDL more specific also makes the Web Service more easily consumed by standard, Web-Service client applications (BizTalk, InfoPath, Word, SharePoint, Excel, AJAX, WebSphere, and many more)!

SAS 9.2 BI Web Services will also support MTOM attachments (Message Transmission Optimization Mechanism—a recent W3C standard). MTOM will be used when a Stored Process generates binary output. Today, most Stored Processes generate Binary Output (because they create reports that contain HTML and GIF images). This is one of the main reasons for having the 'XMLA Web Service' keyword—it means that the Stored Process only generates XML output. MTOM is currently supported by .NET and the Apache Axis 2 Web Service stack. Other Java Web Service stacks such as WebSphere and WebLogic support SOAP With Attachments (SWA), but they will support MTOM in the future.

For more information about SAS BI Web Services, see the *SAS Integration Technologies: Developer's Guide*, which is available at support.sas.com/rnd/itech/library/toc_devguide.html.

WRITING YOUR OWN WEB SERVICE

Even with the availability of SAS BI Web Services, there remain several good reasons for you to create your own custom Web Service. You might want to create your own Web Service because:

- 1) You want to do more than just call SAS. You want some custom mid-tier logic, or you want to call another Web Service and aggregate the information into a single Service.
- 2) You want precise control over the message type (schema), or maybe you have a standard Web Service interface that you need to implement.

SAS provides several choices for writing your own Web Service:

- Call SAS BI Web Services.
- Use the IOM Workspace interface (available for either COM or Java, so you can write your Web Service in either .NET or Java).
- Use the IOM Stored Process Service (terminology note: the 'Stored Process Service' is available as either Java classes or, starting in SAS 9.2, in .NET classes and runs in the client or mid-tier. The 'Stored Process Server' is a server process that actually runs the SAS code for the Stored Process).
- Write a Java or a .NET Web Service that passes messages to SAS through message queues.

MESSAGE QUEUES

As an alternative to Web Services, Message Queues can be very useful in a Service-Oriented Architecture, and they add a level of reliability. Message Queues are also a good way to balance a load across time. The requests can accumulate in the queue, and a server can get to the request when it becomes available. SAS provides two ways to deal with Message Queues: CALL functions and the Package framework. The CALL functions that are typically used to read a message from a queue are: INIT, OPENQUEUE, RECEIVEMESSAGE, CLOSEQUEUE, and TERM. In the Package framework, you can use PACKAGE_PUBLISH with the TO_QUEUE parameter.

Message Queue messages must be mapped appropriately into SAS variables. Therefore, you need to keep in mind the 32k SAS variable limitation in order to avoid truncation.

The spawner was enhanced in SAS 9.1.3 to include "Message Queue Polling" for queues that run under WebSphere MQ. This enhancement enables the spawner to launch SAS when messages arrive in a queue.

PLAIN OLD XML

One example of a fairly common Service that uses plain old XML is Really Simple Syndication (RSS). For a long time, SAS has had the 'URL' access method. Now, there are some services on the Web that will provide stock quotes through RSS. RSS isn't as well suited for getting data as Web Services, but it is another possibility. In SAS code, you could deal with plain old XML by using code similar to the following:

```
filename rssfeed URL "http://somePlaceThatGivesRssFeeds/something.rss";
libname rssfeed xml xmlmap='myMapFromRSS';
proc print data=rssfeed.thetable;
run;
```

THE "...ITY'S"

Security, Scalability, Reliability, and Manageability are the hallmarks of enterprise applications. By virtue of SAS building integration on top of standard platforms, we get the "...ity's".

- Security – SAS BI Web Services for .NET in SAS 9.1 supports the use of the standard Microsoft Web Services Extensions (WSE, pronounced 'WIZZ-EE') WS-Security configuration. For SAS 9.2, both the Java version and the .NET version will support WS-Security through standard configuration. In addition, all versions support standard Transport Layer Security (SSL/TLS) to protect calls to the Web Service.
- Scalability – The SAS Stored Process Server is highly scalable by using Load Balancing. Load Balancing makes it easy to scale out to many Stored Process Servers on many machines.
- Reliability – The Load Balancing implementation detects servers that are down and takes them out-of-use to ensure that your jobs will find a server to run on. An application that is composed of multiple services is susceptible to downtime of any of the services it depends on, therefore, in a SOA, it is especially important to have highly reliable Services.

- Manageability – This is where the use of standard platforms becomes very important. As mentioned earlier, you pick the version (.NET or Java) that your organization prefers to administer and maintain. You use standard tools to maintain the Web Service on the mid-tier. The server-tier (Stored Process Server and Metadata Server) is the same as it would be for a wide variety of other SAS products, which makes it easy for you to maintain one environment that supports many different uses.

THE REST OF THE STORY

SOAP-based Web Services are not the ultimate in distributed computing. The one thing you can be sure of is that new architectural styles (which some might say was in the mainframe 35 years ago) will come along and become a bigger buzz than Service-Oriented Architecture. We are already seeing Representational State Transfer (REST) becoming a buzzword. REST does not require the use of SOAP, but it can use XML, HTTP, and URLs.

CONCLUSION

SAS provides a rich set of tools for your integration toolbox. It's the developer's job to pick the right tool for the job. No one tool is right for every job. At the core of your SAS toolbox is the Stored Process. The Stored Process is highly adaptable to many different uses. You've seen that Stored Processes can be called as Web Services, and they can call Web Services.

Your call to action is to create Stored Processes for your SAS code. Stored Processes open the door for a multitude of applications, which is really the ultimate goal of a SOA—to open the door to more consumers of your service.

REFERENCES

Box, Don. 2004. "A Guide to Developing and Running Connected Systems with Indigo". *MSDN Magazine*.

SAS Institute Inc. 2006. *SAS Integration Technologies*. Available support.sas.com/rnd/itech.

W3C "Web Services Architecture." Available www.w3.org/TR/2002/WD-ws-arch-20021114/#basic.

ACKNOWLEDGMENTS

Thanks to Bryan Allen of SAS for the performance data. The performance tests used the SASHELP.PRDSALE data set with its data replicated to create larger data sets. The binary data transfer was performed by using the SAS OLE/DB provider to call through IOM interfaces to a SAS Workspace. The performance numbers are for reference only, and do not necessarily correspond to what you would see in every situation.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Dan Jahn
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Work Phone: 919 677-8000
E-mail: Dan.Jahn@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.